

# AI Assistant Coding

## Assignment 7.5

Name : GOLI NITHYA HT. No : 2303A52066 Batch: 32

### Task 1 (Mutable Default Argument – Function Bug)

Task: Analyze given code where a mutable default argument causes unexpected behavior. Use AI to fix it.

# Bug: Mutable default argument

```
def add_item(item, items=[]):
    items.append(item)
    return items
print(add_item(1))
print(add_item(2))
```

Expected Output: Corrected function avoids shared list bug.

#### Corrected Code:

```
def add_item(item, items=None):
    if items is None:
        items = []
    items.append(item)
    return items
print(add_item(1))
print(add_item(2,[10,20]))
print(add_item(3))
```

#### Output:

```
[1]
[10, 20, 2]
[3]
```

---

### Task 2 (Floating-Point Precision Error)

Task: Analyze given code where floating-point comparison fails.

Use AI to correct with tolerance.

# Bug: Floating point precision issue

```
def check_sum():
```

```
    return (0.1 + 0.2) == 0.3
print(check_sum())
```

Expected Output: Corrected function

### Corrected Code:

```
# find the error in the code and fix it
def check_sum():
    return abs((0.1 + 0.2) - 0.3) < 1e-9
print(check_sum())
```

### Output:

```
True
```

---

### Task 3 (Recursion Error – Missing Base Case)

Task: Analyze given code where recursion runs infinitely due to missing base case. Use AI to fix.

# Bug: No base case

```
def countdown(n):
    print(n)
    return countdown(n-1)
countdown(5)
```

Expected Output : Correct recursion with stopping condition.

### Corrected Code:

```
def countdown(n):
    print(n)
    if n > 0:
        return countdown(n-1)
countdown(5)
```

### Output:

```
5
4
3
2
1
0
```

---

### Task 4 (Dictionary Key Error)

Task: Analyze given code where a missing dictionary key causes error.  
Use AI to fix it.

# Bug: Accessing non-existing key

```
def get_value():
    data = {"a": 1, "b": 2}
    return data["c"]
print(get_value())
```

Expected Output: Corrected with .get() or error handling.

**Corrected Code:**

```
def get_value():
    data = {"a": 1, "b": 2}
    return data.get("c", "Key not found")
print(get_value())
```

**Output:**

```
Key not found
```

---

## Task 5 (Infinite Loop – Wrong Condition)

Task: Analyze given code where loop never ends. Use AI to detect and fix it.

# Bug: Infinite loop

```
def loop_example():
    i = 0
    while i < 5:
        print(i)
```

Expected Output: Corrected loop increments i.

**Corrected Code:**

```
def loop_example():
    i = 0
    while i < 5:
        print(i)
        i += 1
loop_example()
```

**Output:**

```
0
```

```
1  
2  
3  
4
```

---

## Task 6 (Unpacking Error – Wrong Variables)

Task: Analyze given code where tuple unpacking fails. Use AI to fix it.

```
# Bug: Wrong unpacking
```

```
a, b = (1, 2, 3)
```

Expected Output: Correct unpacking or using `_` for extra values.

### Corrected Code:

```
a, b, c = (1, 2, 3)  
print(a, b, c)
```

### Output:

```
1 2 3
```

---

## Task 7 (Mixed Indentation – Tabs vs Spaces)

Task: Analyze given code where mixed indentation breaks execution. Use AI to fix it.

```
# Bug: Mixed indentation
```

```
def func():  
    x = 5  
        y = 10  
    return x+y
```

Expected Output : Consistent indentation applied.

### Corrected Code:

```
def func():  
    x = 5  
    y = 10  
    return x+y  
result = func()  
print(result)
```

## **Output:**

```
15
```

---

## **Task 8 (Import Error – Wrong Module Usage)**

Task: Analyze given code with incorrect import. Use AI to fix.

# Bug: Wrong import

```
import maths  
print(maths.sqrt(16))
```

Expected Output: Corrected to import math

## **Corrected Code:**

```
import math  
print(math.sqrt(16))
```

## **Output:**

```
4.0
```

---

## **Task 9 (Unreachable Code – Return Inside Loop)**

Task: Analyze given code where a return inside a loop prevents full iteration. Use AI to fix it.

# Bug: Early return inside loop

```
def total(numbers):  
    for n in numbers:  
        return n  
print(total([1,2,3]))
```

Expected Output: Corrected code accumulates sum and returns after loop.

## **Corrected Code:**

```
def total(numbers):  
    total_sum = 0  
    for n in numbers:  
        total_sum += n  
    return total_sum  
print(total([1,2,3]))
```

## **Output:**

6

---

### **Task 10 (Name Error – Undefined Variable)**

Task: Analyze given code where a variable is used before being defined.  
Let AI detect and fix the error.

# Bug: Using undefined variable

```
def calculate_area():
    return length * width
print(calculate_area())
```

Requirements:

- Run the code to observe the error.
- Ask AI to identify the missing variable definition.
- Fix the bug by defining length and width as parameters.
- Add 3 assert test cases for correctness.

Expected Output :

- Corrected code with parameters.
- AI explanation of the bug.

Successful execution of assertions.

## **Corrected Code:**

```
def calculate_area(length, width):
    return length * width
print(calculate_area(5, 10))
print(calculate_area(7, 3))
print(calculate_area(0, 5))
```

## **Output:**

50  
21  
0

## **Explanation:**

This code will not work because the function `calculate\_area` is defined without any parameters, but it tries to use the variables `length` and `width` which are not defined within the function. To fix this, you need to define `length` and `width` as parameters of the function so that you can pass values to it when calling the function.

---

### Task 11 (Type Error – Mixing Data Types Incorrectly)

Task: Analyze given code where integers and strings are added incorrectly. Let AI detect and fix the error.

# Bug: Adding integer and string

```
def add_values():
    return 5 + "10"
print(add_values())
```

Requirements:

- Run the code to observe the error.
- AI should explain why int + str is invalid.
- Fix the code by type conversion (e.g., int("10") or str(5)).
- Verify with 3 assert cases.

Expected Output #6:

- Corrected code with type handling.
- AI explanation of the fix.

Successful test validation.

### Corrected Code:

```
def add_values():
    return 5 + int("10")
print(add_values())
print(add_values())
print(add_values())
```

Output:

```
15
15
15
```

Explanation:

The code will not work because it is trying to add an integer (5) and a string ("10"), which is not allowed in Python. This will raise a `TypeError`. To fix this, you can either convert the string to an integer or the integer to a string before performing the addition.

---

### **Task 12 (Type Error – String + List Concatenation)**

Task: Analyze code where a string is incorrectly added to a list.

# Bug: Adding string and list

```
def combine():
    return "Numbers: " + [1, 2, 3]
print(combine())
```

Requirements:

- Run the code to observe the error.
- Explain why `str + list` is invalid.
- Fix using conversion (`str([1,2,3])` or `" ".join()`).
- Verify with 3 assert cases.

Expected Output:

- Corrected code
- Explanation
- Successful test validation

**Corrected Code:**

```
def combine():
    return "Numbers: " + str([1, 2, 3])
print(combine())
print(combine())
print(combine())
```

**Output:**

```
Numbers: [1, 2, 3]
Numbers: [1, 2, 3]
Numbers: [1, 2, 3]
```

**Explanation:**

The code will not work because it is trying to concatenate a string ("Numbers: ") with a list ([1, 2, 3]). In Python, you cannot directly concatenate a string and a list, which will result in a `TypeError`.

---

### **Task 13 (Type Error – Multiplying String by Float)**

Task: Detect and fix code where a string is multiplied by a float.

```
# Bug: Multiplying string by float
```

```
def repeat_text():
    return "Hello" * 2.5
print(repeat_text())
```

**Requirements:**

- Observe the error.
- Explain why float multiplication is invalid for strings.
- Fix by converting float to int.
- Add 3 assert test cases.

**Corrected Code:**

```
def repeat_text():
    return "Hello" * 2
print(repeat_text())
print(repeat_text())
print(repeat_text())
```

**Output:**

```
HelloHello
HelloHello
HelloHello
```

**Explanation:**

The code will not work because you cannot multiply a string by a non-integer (in this case, 2.5). In Python, multiplying a string by an integer repeats the string that many times, but multiplying by a float is not allowed and will raise a `TypeError`.

---

## Task 14 (Type Error – Adding None to Integer)

Task: Analyze code where None is added to an integer.

# Bug: Adding None and integer

```
def compute():
    value = None
    return value + 10
print(compute())
```

Requirements:

- Run and identify the error.
- Explain why NoneType cannot be added.
- Fix by assigning a default value.
- Validate using asserts.

### Corrected Code:

```
def compute(value=0):
    return value + 10
print(compute())
print(compute(5))
```

### Output:

```
10
15
```

### Explanation:

The code will not work because the variable `value` is set to `None`, and you cannot add an integer to `None`. To fix this, you should initialize `value` with a numeric value, such as 0, or modify the function to accept a parameter with a default value.

---

## Task 15 (Type Error – Input Treated as String Instead of Number)

Task: Fix code where user input is not converted properly.

# Bug: Input remains string

```
def sum_two_numbers():
```

```
a = input("Enter first number: ")
b = input("Enter second number: ")
return a + b
print(sum_two_numbers())
```

### Requirements:

- Explain why `input` is always string.
- Fix using `int()` conversion.
- Verify with assert test cases.

### Corrected Code:

```
def sum_two_numbers():
    a = input("Enter first number: ")
    b = input("Enter second number: ")
    return int(a) + int(b)
print(sum_two_numbers())
```

### Output:

```
Enter first number: 5
Enter second number: 6
11
```

### Explanation:

This code will not work as intended because the `input()` function in Python always returns a string. When you use the `+` operator with two strings, it performs string concatenation instead of numerical addition. For example, if you enter `'3'` and `'4'`, the output will be `'34'` instead of `7`.